

06/30/00 1c715 U.S. PTO

Please type a plus sign (+) inside this box → ☐

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No. 2207/8609

First Inventor or Application Identifier Stephan JOURDAN et al

Title TRACE INDEXING VIA TRACE END ADDRESSES

Express Mail Label No.

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

- ☒ * Fee Transmittal Form (e.g., PTO/SB/17)
(Submit an original and a duplicate for fee processing)
- ☒ Specification [Total Pages 11] 1
(preferred arrangement set forth below)
 - Descriptive title of the Invention
 - Cross References to Related Applications
 - Statement Regarding Fed sponsored R & D
 - Reference to Microfiche Appendix
 - Background of the Invention
 - Brief Summary of the Invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claim(s)
 - Abstract of the Disclosure
- ☒ Drawing(s) (35 U.S.C. 113) [Total Sheets 3] 1
- Oath or Declaration [Total Pages 1] 1
 - ☐ Newly executed (original or copy)
 - ☐ Copy from a prior application (37 C.F.R. § 1.63(d))
(for continuation/divisional with Box 16 completed)
 - ☐ DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

* NOTE FOR ITEMS 1 & 13 IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

- ☐ Microfiche Computer Program (Appendix)
- Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
 - ☐ Computer Readable Copy
 - ☐ Paper Copy (identical to computer copy)
 - ☐ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

- ☐ Assignment Papers (cover sheet & document(s))
- ☐ 37 C.F.R. § 3.73(b) Statement ☐ Power of Attorney
(when there is an assignee)
- ☐ English Translation Document (if applicable)
- ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
- ☐ Preliminary Amendment
- ☒ Return Receipt Postcard (MPEP 503)
(Should be specifically itemized)
- * Small Entity ☐ Statement filed in prior application
Statement(s) ☐ Status still proper and desired
(PTO/SB/09-12)
- ☐ Certified Copy of Priority Document(s)
(if foreign priority is claimed)
- ☐ Other:

16. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment.

☒ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No. /

Prior application information: Examiner Group / Art Unit:

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

☒ Customer Number or Bar Code Label



(Insert Customer No. or attach bar code label here)

or ☐ Correspondence address below

Name	23838		
PATENT TRADEMARK OFFICE			
Address			
City	State	Zip Code	
Country	Telephone	Fax	

Name (Print/Type)	Robert L. Hails, Jr.	Registration No. (Attorney/Agent)	39,702
Signature		Date	6/30/2000

Burden Hour Statement. This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

A PATENT APPLICATION FOR

TRACE INDEXING VIA TRACE END ADDRESSES

INVENTORS:

Stephan Jourdan
Lihu Rappoport
Ronny Ronen
Adi Yoaz
of Intel Corporation

PREPARED BY:

Robert L. Hails Jr.
KENYON & KENYON
1500 K Street, N.W.
Washington, DC 20005
(202) 220-4200

00000000-42300000

TRACE INDEXING VIA TRACE END ADDRESSES

BACKGROUND

The present invention provides a new trace assembly paradigm for processing circuits.

5 FIG. 1 is a block diagram illustrating the process of program execution in a conventional processor. Program execution may include three stages: front end 110, execution 120 and memory 130. The front-end stage 110 performs instruction pre-processing. Front end processing is designed with the goal of supplying valid decoded instructions to an execution core with low latency and high bandwidth. Front-end
10 processing can include instruction prediction, decoding and renaming.

As the name implies, the execution stage 120 performs instruction execution. The execution stage 120 typically communicates with a memory 130 to operate upon data stored therein.

15 The front end stage 110 may include a trace cache (not shown) to reduce the latency of instruction decoding and to increase front end bandwidth. A trace cache is a circuit that assembles sequences of dynamically executed instructions into logical units called "traces." The program instructions may have been assembled into a trace from non-contiguous regions of an external memory space but, when they are assembled in a trace, the instructions appear in program order. Typically, a trace may begin with an
20 instruction of any type. The trace may end when one of number of predetermined trace end conditions occurs, such as a trace size limit, a maximum number of conditional branches occurs or an indirect branch or return instruction occurs.

25 Prior art traces are defined by an architecture having a single entry point but possibly many exit points. This architecture, however, causes traces to exhibit instruction redundancy. Consider, by way of example, the following code sequence:

```
    If (cond)
        A
    B
```

This simple code sequence produces two possible traces: 1) B and 2) AB. When assembled in a trace cache, each trace may be stored independently of the other. Thus, the B instruction would be stored twice in the trace cache. This redundancy lowers system performance. Further, because traces may start on any instruction, the B instruction also may be recorded in multiple traces due to instruction alignment discrepancies that may occur. This instruction redundancy limits the bandwidth of front-end processing.

Accordingly, there is a need in the art for a front-end stage that reduces instruction redundancy in traces.

10 **BRIEF DESCRIPTION OF THE DRAWINGS**

FIG. 1 is a block diagram illustrating functional processing in program execution.

FIG. 2 is a block diagram of a front-end stage 200 according to an embodiment of the present invention.

15 FIG. 3 is a flow diagram illustrating operation 1000 of the XFU 260 according to an embodiment of the invention.

FIGS. 4(a)-4(c) illustrate construction of extended blocks according to embodiments of the present invention.

FIG. 5 illustrates construction of an extended block according to an embodiment of the present invention.

20 FIG. 6 illustrates construction of a complex extended block according to an embodiment of the present invention.

DETAILED DESCRIPTION

Embodiments of the present invention assemble a new type of traces, called “extended blocks” herein, according to an architecture that permits several entry points
25 but only a single exit point. These extended blocks may be indexed based upon the

address of the last instruction therein. The extended block architecture provides several advantages over prior architectures:

- instruction redundancies can be eliminated,
- multiple entry points are permitted,
- extended blocks may be extended dynamically, and
- basic blocks may be shared among various extended blocks.

FIG. 2 is a block diagram of a front-end stage 200 according to an embodiment of the present invention. The front end 200 may include an instruction cache 210 and an extended block cache ("XBC") 220. The instruction cache 210 may be based on any number of known architectures for front-end systems. Typically, they include an instruction memory 230, a branch predictor 240 and an instruction decoder 250. Program instructions may be stored in the cache memory 230 and indexed by an instruction pointer. Instructions may be retrieved from the cache memory 230, decoded by the instruction decoder 250 and passed to the execution unit (not shown). The branch predictor 240 may assist in the selection of instructions to be retrieved from the cache memory 230 for execution. As is known, instructions may be indexed by an address, called an "instruction pointer" or "IP."

According to an embodiment, an XBC 220 may include a fill unit ("XFU") 260, a block predictor ("XBTB") 270 and a block cache 280. The XFU 260 may build the extended blocks. The block cache 280 may store the extended blocks. The XBTB 270 may predict which extended blocks, if any, are likely to be executed and may cause the block cache to furnish any predicted blocks to the execution unit. The XBTB 270 may store masks associated with each of the extended blocks stored by the block cache 280, indexed by the IP of the terminal instruction of the extended blocks.

The XBC 220 may receive decoded instructions from the instruction cache 210. The XBC 220 also may pass decoded instructions to the execution unit (not shown). A selector 290 may select which front-end source, either the instruction cache 210 or the XBC 220, will supply instructions to the execution unit. In an embodiment, the block cache 280 may control the selector 290.

As discussed, the block cache 280 may be a memory that stores extended blocks. According to an embodiment, the extended blocks may be multiple-entry, single-exit traces. An extended block may include a sequence of program instructions. It may terminate in a conditional branch, an indirect branch or based upon a predetermined
5 termination condition such as a size limit. Again, the block cache 280 may index the extended blocks based upon an IP of the terminal instruction in the block.

Extended blocks are useful because, whenever program flow enters the extended block, the flow necessarily progresses to the terminal instruction in the extended block. An extended block may contain any conditional or indirect branches only as a terminal
10 instruction. Thus, the extended block may have multiple entry points. According to an embodiment, an unconditional branch need not terminate an extended block.

According to an embodiment, a hit/miss indication from the block cache 280 may control the selector 290.

FIG. 3 is a flow diagram illustrating operation 1000 of the XBC 220 according to
15 an embodiment of the invention. Operation may begin when the XBC 220 determines whether an IP of a terminal instruction from an extended block may be predicted (Stage 1010). If so, the XBC 200 performs such a prediction (Stage 1020). The prediction may be made by the XBTB 220. Based on the IP of the predicted terminal instruction, the block cache 280 may indicate a “hit” or a “miss” (Stage 1030). A hit indicates that the
20 block cache 280 stores an extended block that terminates at the predicted IP. In this case, the XFU 260 may cause an extended block to be retrieved from the block cache 280 and forwarded to the execution units (Stage 1040). Thereafter, the process may return to Stage 1010 and repeat.

If the predicted IP does not hit the block cache 280 or if an IP of a terminal
25 instruction could not be predicted at Stage 1010, the XFU 260 may build a new extended block. Decoded instructions from the instruction cache may be forwarded to the execution unit (Stage 1050). The XFU 260 also may receive the retrieved instructions from the instruction cache system 210. It stores instructions in a new block until it reaches a terminal condition, such as a conditional or implicit branch or a size limitation

(Stage 1060). Having assembled a new block, the XFU 260 may determine how to store it in the block cache 280.

The XFU 260 may determine whether the terminal IP of the new block hits the block cache (Stage 1070). If not, then the XFU 260 simply may cause the new block to be stored in the block cache 280 (Stage 1080).

If the IP hits the block cache, then the XFU 260 may compare the contents of the new block with the contents of an older block stored in the block cache that generated the hit. The XFU 260 may determine whether the contents of the new block are subsumed entirely within the old block (Stage 1090). If so, the XFU 260 need not store the new block in the block cache 280 because it is present already in the old block. If not, the XFU 260 may determine whether the contents of the old block are subsumed within the new block (Stage 1100). If so, the XFU 260 may write the new block over the old block in the cache (Stage 1110). This has the effect of extending the old block to include the new block.

If neither of the above conditions is met, then the new block and the old block may be only partially co-extensive. There are several alternatives for this case. In a first embodiment, the XFU 260 may store the non-overlapping portion of the new block in the block cache 280 as a separate extended block (Stage 1120). Alternatively, the XFU 260 may create a complex extended block from the new block and the old block (Stage 1130). These are discussed in greater detail below.

Once the new block is stored in the block cache 280, at the conclusion of Stages 1110, 1120 or 1130, the XBTB may be updated to reflect the contents of the block cache 280 (Stage 1140). Thereafter, operation may return to Stage 1010 for a subsequent iteration.

FIGS. 4(a)-4(c) schematically illustrate the different scenarios that may occur if the IP pointer of the new extended block XB_{new} matches that of an extended block stored previously within the block cache 280 (XB_{old}). In FIG. 4(a), the older extended block XB_{old} is co-extensive with the new extended block XB_{new} but is longer. In this case, the new extended block XB_{new} should not be stored separately within the block cache 280;

the older extended block may be used instead. In FIG. 4(a), the older extended block XB_{old} is co-extensive with a portion of the new extended XB_{new} but XB_{new} is longer. In this case, the older extended block may be extended to include the additional instructions found in XB_{new} .

5 In a third case, shown in FIG. 4(c), only a portion of XB_{new} and XB_{old} are co-extensive. In this case, a “suffix” of each extended block coincides but the two extended blocks have different “prefixes.” In a first embodiment, the XFU 260 may store a prefix of the new extended block as an extended block all its own. The prefix may end in an unconditional jump pointing to an appropriate location in the older extended block. This
10 solution is shown in FIG. 5.

Alternatively, the XFU 260 may assemble a single, complex extended block merging the two extended blocks. In this embodiment, the XFU 260 may extend the older extended block by adding the prefix of the new extended block to a head of the older extended block. The prefix of the new extended block may conclude in an
15 unconditional jump pointing to the common suffix. This solution is shown in FIG. 6. This embodiment creates a single, longer extended block instead of two short extended blocks and thereby contributes to increased bandwidth. In such an embodiment, the XFU 260 may generate a mask in the block cache 280 that is associated with the complex extended block.

20 According to an embodiment, the XBTB 270 may predict extended blocks to be used.

Returning to FIG. 2, the XBTB 270 may predict extended blocks for use during program execution. The XBTB 270 may store information for each of the extended blocks stored in the block cache 280. In an embodiment, the XBTB 270 may store masks for
25 each block cache. These masks may identify whether a corresponding extended block is a complex or non-complex block (compare FIGS. 4(a) and 4(b) with FIG. 6). For non-complex extended blocks, the XBTB 270 may store a mask identifying the length of the extended blocks. For complex extended blocks, the XBTB 270 may store a mask that distinguishes multiple prefixes from each other. Thus, while the prefixes may be linearly
30 appended to each other as shown in FIG. 6, a mask as stored in the XBTB permits the

XBTB to determine the position of an entry point to an extended block based on an instruction's IP.

As described above, an XBTB 270 may store a mapping among blocks. As noted, the XBTB 270 may identify the terminal IP of each block and may store masks for each extended block. The XBTB 270 also may store a mapping identifying links from one XBTB to another. For example, if a first extended block ended in a conditional branch, program execution may reach a second extended block by following one of the directions of the terminal branch and execution may reach a third extended block by following the other direction. The XBTB 270 may store a mapping of both blocks. In this embodiment, the XFU 260 may interrogate the XBTB 270 with an IP. By way of response, the XBC 220 may respond with a hit or miss indication and, if the response is a hit, may identify one or more extended blocks in the block cache to which the IP may refer. In this embodiment, the XBC 220 may determine whether a terminal IP may be predicted and whether the predicted address hits the cache (Stages 1010, 1030).

According to an embodiment, XBC bandwidth may be improved by using conditional branch promotion for terminal instructions. As is known, conditional branch promotion permits a conditional branch to be treated as an unconditional branch if it is found that the branch is nearly monotonic. For example, if during operation, it is determined that program execution at a particular branch tends to follow one of the branches 99% of the time, the branch may be treated as an unconditional jump. In this case, two extended blocks may be joined as a single extended block. This embodiment further contributes to XBC bandwidth.

According to another embodiment, XBC latency may be minimized by having the block cache 280 store extended block instructions in decoded form. Thus, the instructions may be stored as microinstructions (commonly, "uops").

Several embodiments of the present invention are specifically illustrated and described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.

WE CLAIM:

1. A trace having a multiple-entry, single exit architecture.
2. The trace of claim 1, wherein the trace is a complex trace having multiple independent prefixes and a common, shared suffix.
3. The trace of claim 1, wherein the trace is indexed by an address of a terminal instruction therein.
4. A front-end system for a processor, comprising:
 - an instruction cache system,
 - an extended block cache system, comprising:
 - a fill unit provided in communication with the instruction cache system,
 - a block cache, and
 - a selector coupled to the output of the instruction cache system and to an output of the block cache.
5. The front-end system of claim 4, wherein the extended block cache system further comprises a block predictor provided in communication with the fill unit and the block cache.
6. The front-end system of claim 4, wherein the block cache is to store traces having a multiple-entry, single exit architecture.
7. The front-end system of claim 4, wherein the block cache is to store complex traces having multiple independent prefixes and a common suffix.
8. The front-end system of claim 7, wherein the extended block cache system further comprises a block predictor to store masks associated with the complex traces, the masks distinguishing the prefixes from each other.
9. A method of managing extended blocks, comprising:
 - predicting an address of a terminal instruction of an extended block to be used,

determining whether the predicted address matches an address of a terminal instruction of a previously created extended block, and
selecting one of the extended block in the event of a match.

10. The method of claim 9, further comprising creating a new extended block when
5 there is no match.

11. The method of claim 10, wherein the creating comprises:
receiving new instructions until a terminal condition occurs,
assembling the new instructions into an extended block,
determining whether an address of a terminal instruction in the new block
10 matches an address of a terminal instruction of a pre-existing block, and
unless a match occurs, storing the new block in a memory.

12. The method of claim 11, wherein the storing comprises, when an older block causes a match, storing the new block over the old block in a memory if the old block is subsumed within the new block.

13. The method of claim 11, wherein the storing comprises, when an older block causes a match, dropping the new block if the new block is subsumed within the older block.

14. The method of claim 11, wherein the storing comprises, when an older block causes a match, creating a complex block if the new block and the older block share a
20 common suffix but include different prefixes.

15. The method of claim 9, further comprising outputting instructions of the selected block for execution.

16. A processing engine, comprising:
a front end stage to store multiple-entry, single exit traces, and
25 an execution unit in communication with the front end stage.

17. The processing engine of claim 16, wherein the front-end stage comprises:
an instruction cache system,
an extended block cache system, comprising:

a fill unit provided in communication with the instruction cache system,
a block cache, and

a selector coupled to the output of the instruction cache system and to an output
of the block cache.

5 18. The processing engine of claim 17, wherein the block cache is to store the
multiple-entry, single exit traces.

19. The processing engine of claim 17, wherein the extended block cache system
further comprises a block predictor provided in communication with the fill unit and the
block cache.

ABSTRACT

A new class traces for a processing engine, called "extended blocks," possess an architecture that permits possible many entry points but only a single exit point. These extended blocks may be indexed based upon the address of the last instruction therein.

- 5 Use of the new trace architecture provides several advantages, including reduction of instruction redundancies, dynamic block extension and a sharing of instructions among various extended blocks.

FIG. 1

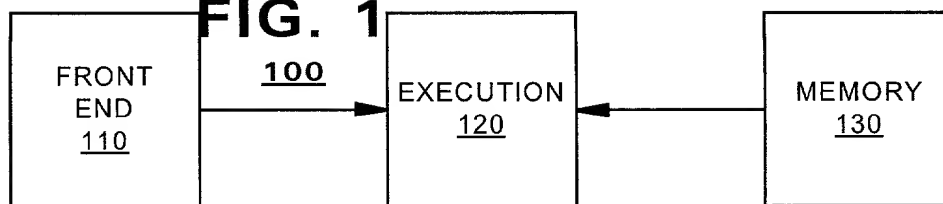
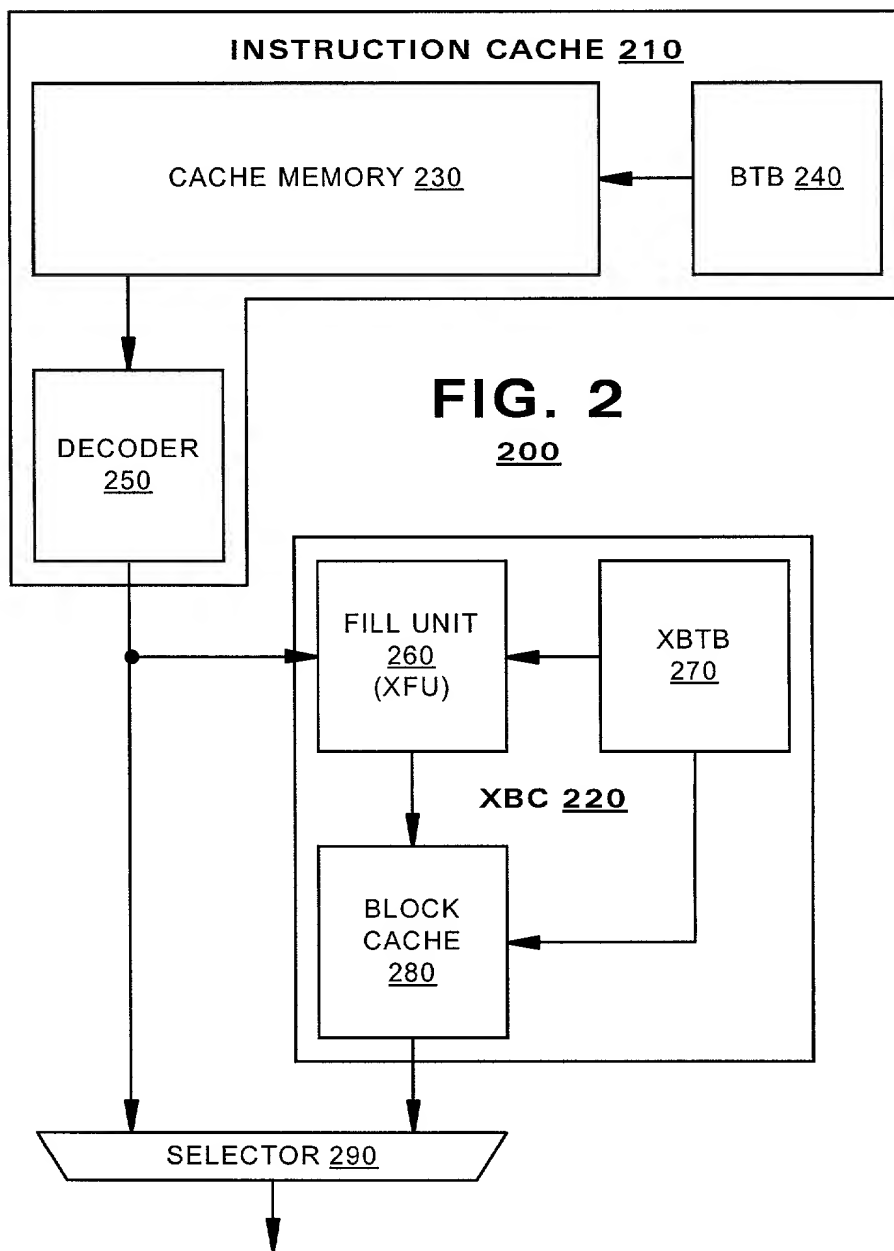


FIG. 2
200



```

graph TD
    1010{{CAN TERMINAL IP ADDRESS BE PREDICTED? 1010}}
    1020[PREDICT TERMINAL IP ADDRESS OF NEXT BLOCK 1020]
    1030{{DOES PREDICTED ADDRESS HIT CACHE? 1030}}
    1040[RETRIEVE BLOCK AND PASS TO EXECUTION 1040]
    1050[RETRIEVE INST. FROM INST. CACHE 1050]
    1060[BUILD NEW BLOCK; PASS TO EXECUTION 1060]
    1070{{DOES ADDRESS OF TERM. INST. HIT CACHE? 1070}}
    1080[STORE NEW BLOCK IN CACHE 1080]
    1090{{IS NEW BLOCK SUBSUMED WITHIN OLD BLOCK? 1090}}
    1100{{IS OLD BLOCK SUBSUMED WITHIN NEW BLOCK? 1100}}
    1110[EXTEND OLD BLOCK TO INCLUDE REMAINDER OF NEW BLOCK 1110]
    1120[STORE PORTION OF NEW BLOCK IN CACHE 1120]
    1130[CREATE COMPLEX BLOCK 1130]
    1140[UPDATE XBTB 1140]

    1010 -- YES --> 1020
    1010 -- NO --> 1050
    1020 --> 1030
    1030 -- YES --> 1040
    1030 -- NO --> 1050
    1040 --> 1140
    1050 --> 1060
    1060 --> 1070
    1070 -- NO --> 1080
    1070 -- YES --> 1090
    1080 --> 1140
    1090 -- YES --> 1140
    1090 -- NO --> 1100
    1100 -- YES --> 1110
    1100 -- NO --> 1120
    1110 --> 1140
    1120 --> 1140
    1120 -.-> 1130
    1130 --> 1140
    1140 --> 1010

```

FIG. 3
1000

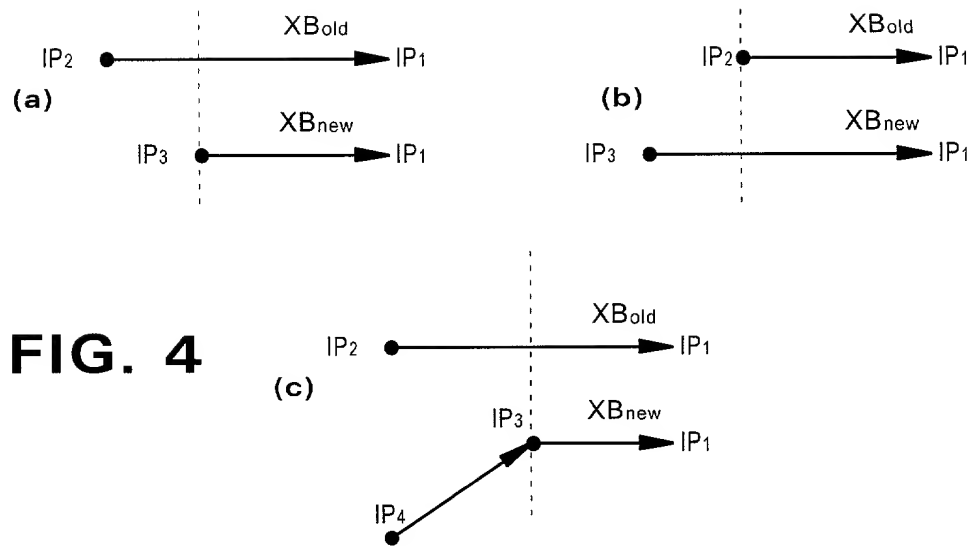


FIG. 4

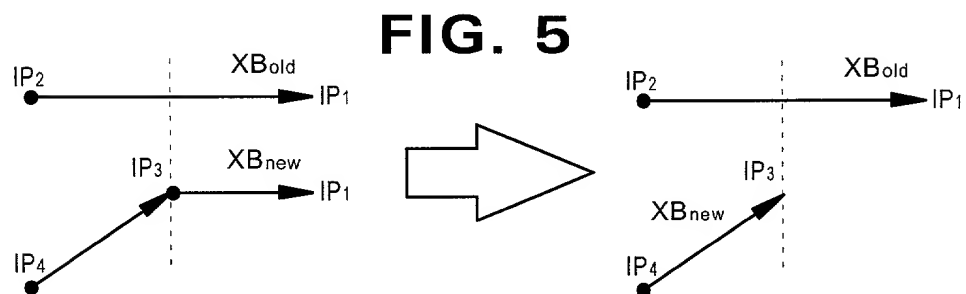


FIG. 5

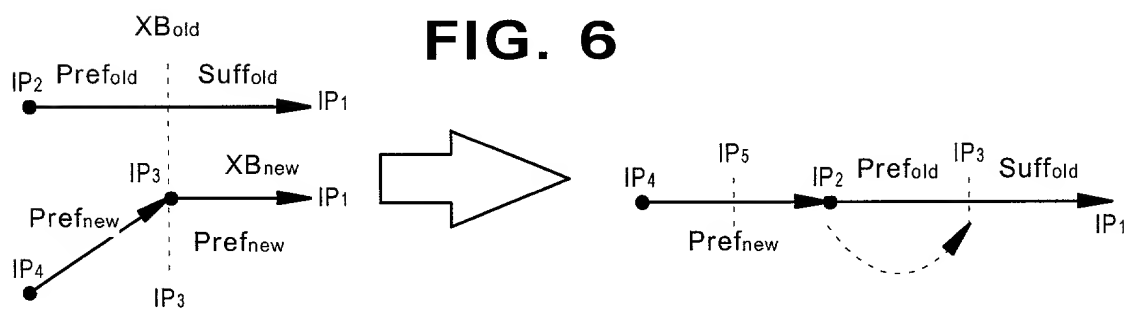


FIG. 6